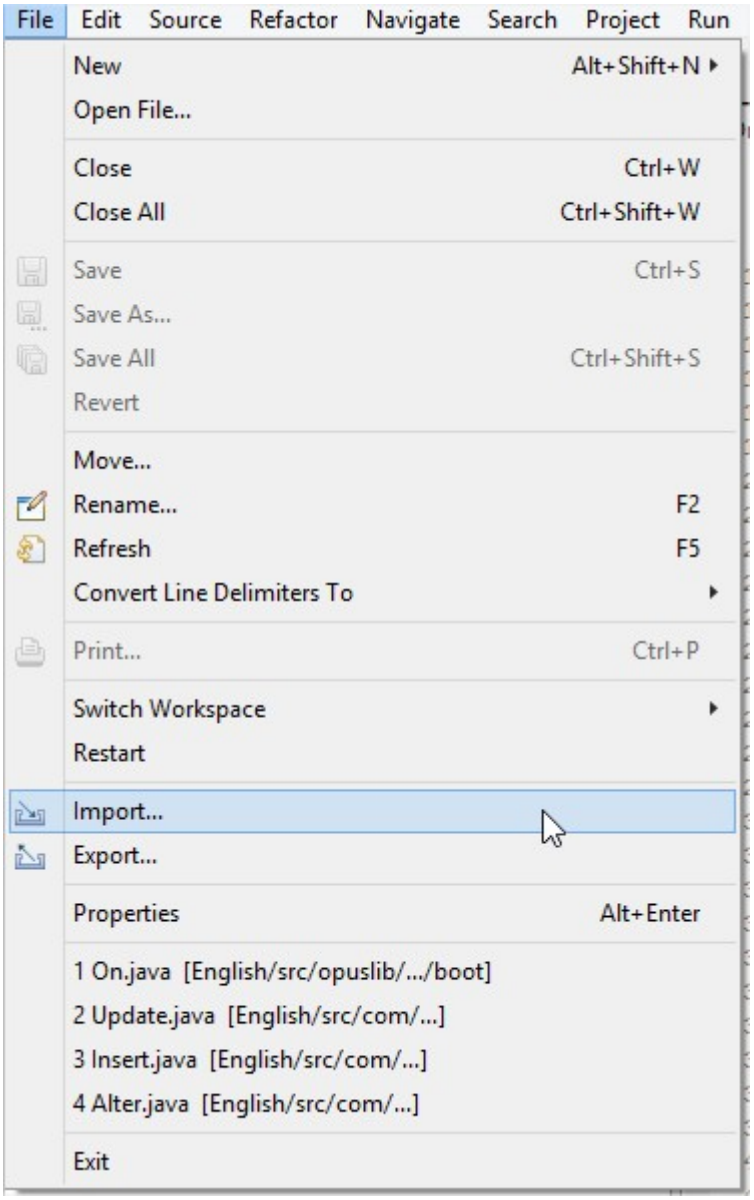
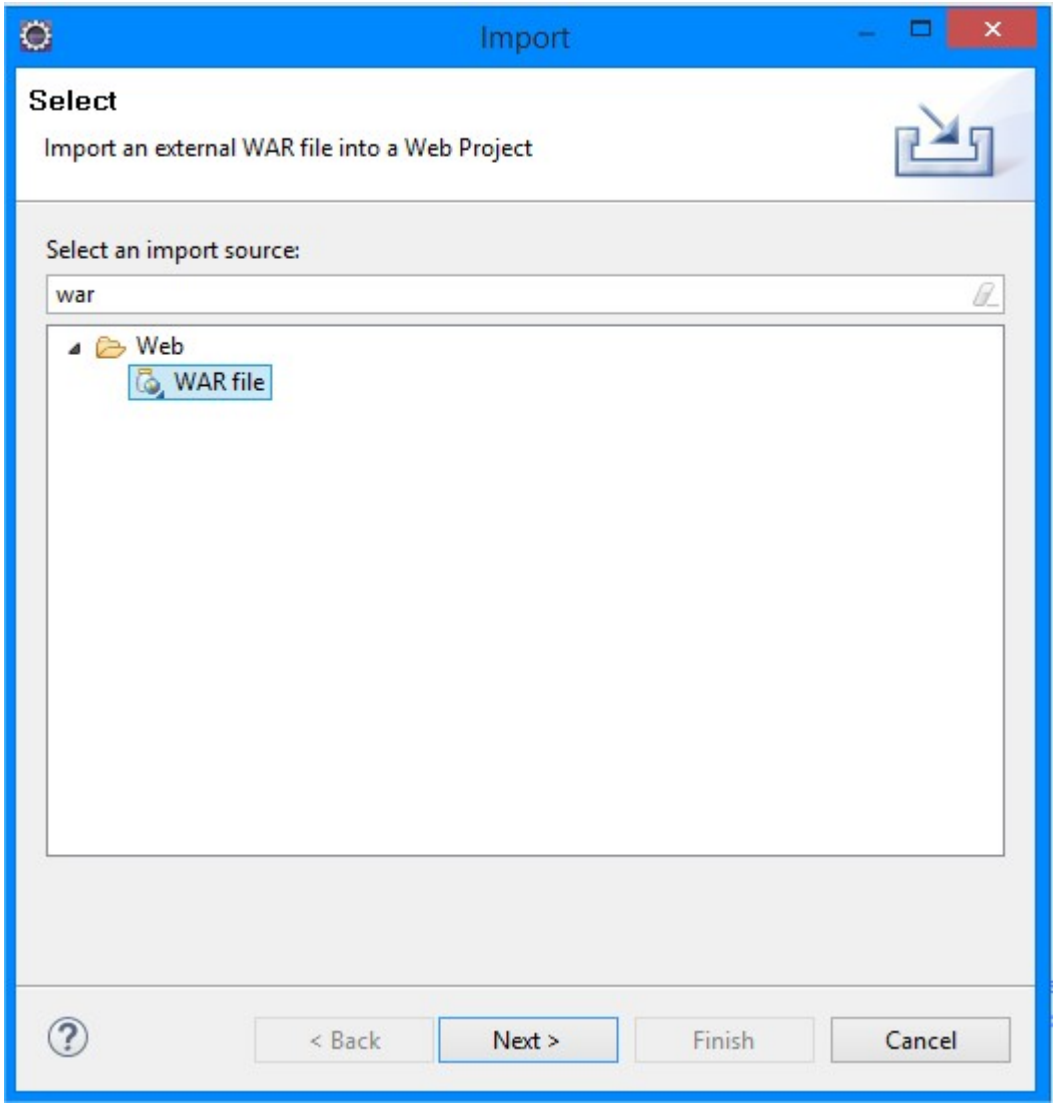
Documentation module (Coeur open-source)

Importation du module Opuslib-base-module dans Eclipse

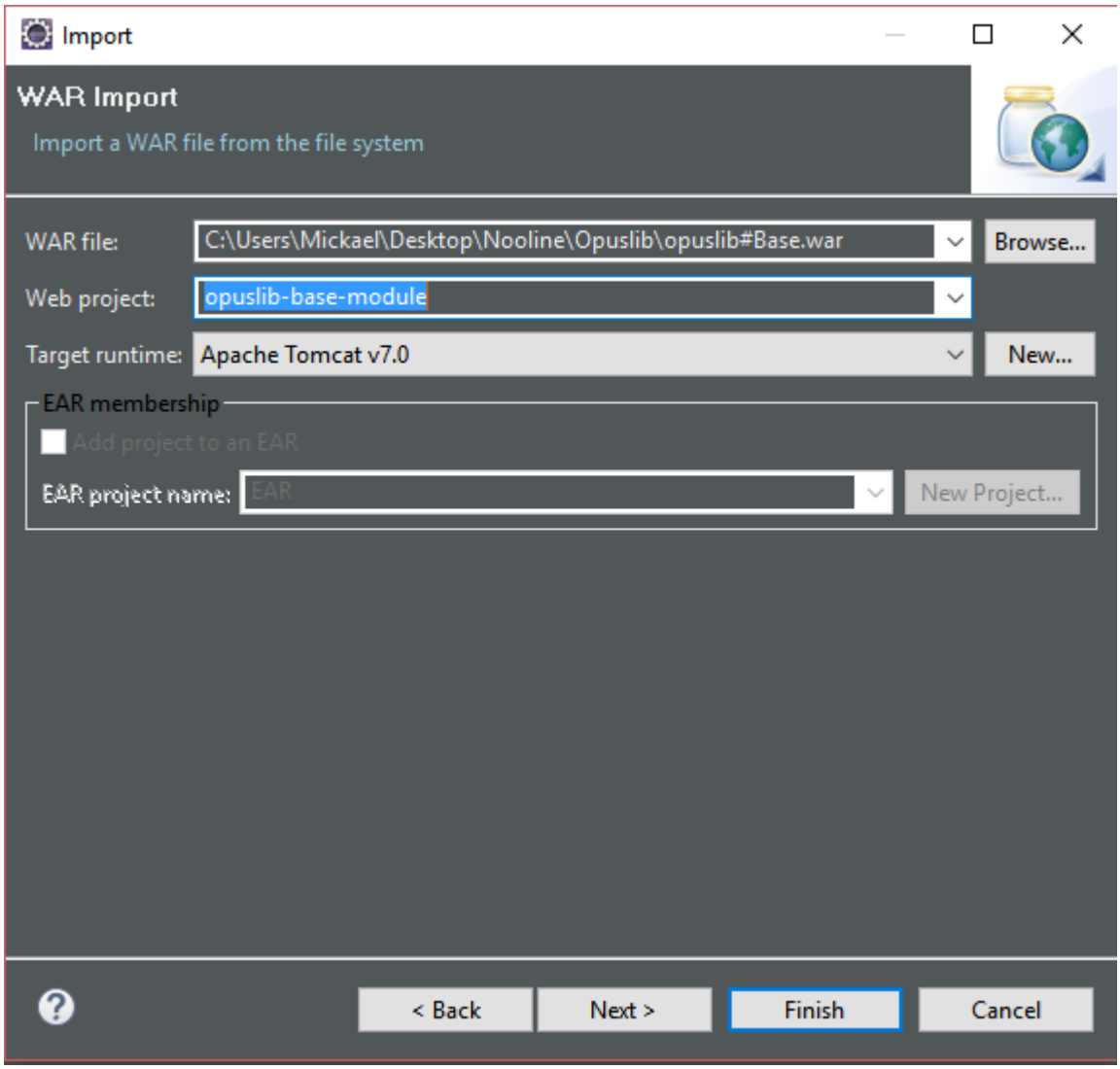
1. Etape 1



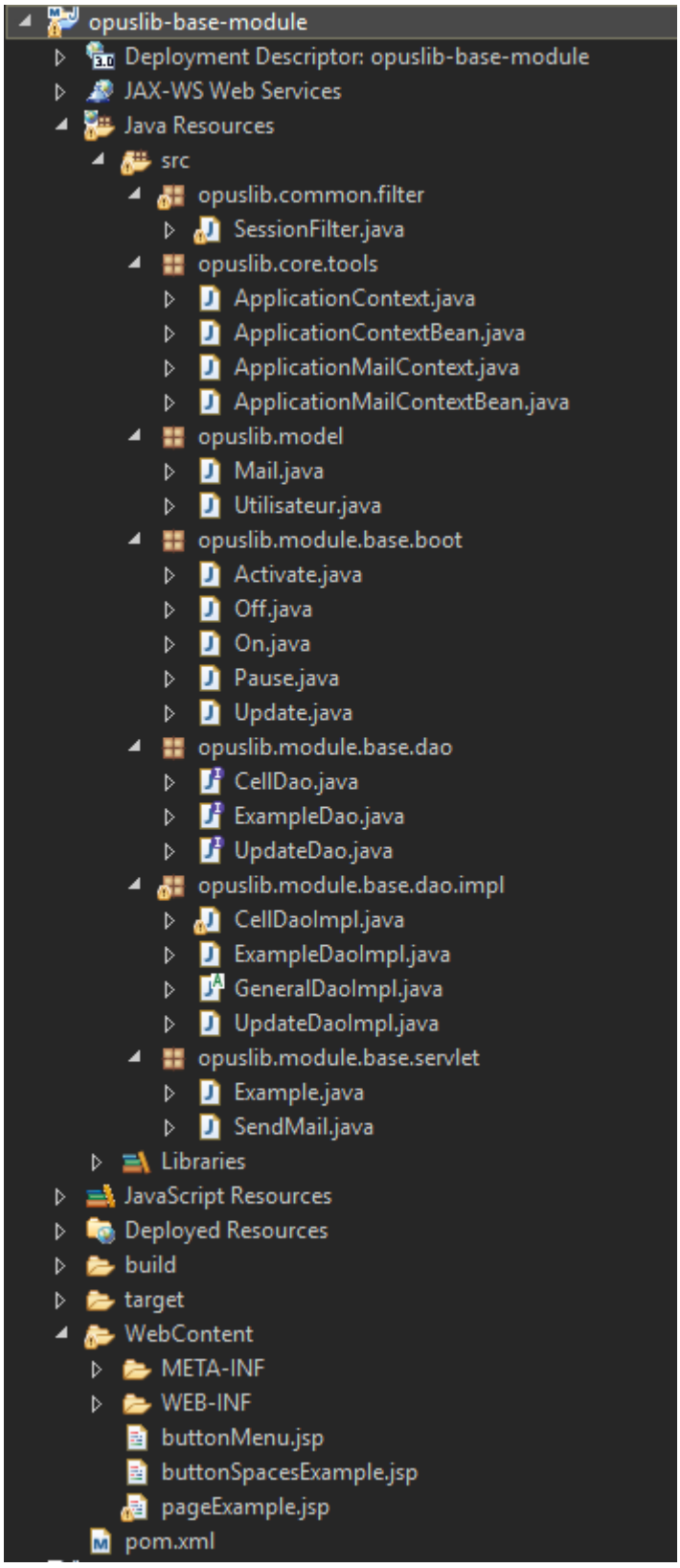
2. Etape 2


Import
Select
Import an external WAR file into a Web Project
Select an import source:
war
Web
WAR file
< Back
Next >
Finish
Cancel

3. Etape 3



Architecture d'un module



- **com.opuslib.common.filter Obligatoire.**

Contient le filtre qui permet de récupérer l'id de l'utilisateur connecté ainsi que toute la map de langue.

- **opuslib.module.base.boot Obligatoire.**

Contient les différentes servlets qui gèrent l'activation, la pause, la mise à jour, l'ajout et la suppression d'un module.

- **opuslib.module.base.dao et opuslib.module.base.dao.impl Obligatoire sauf ExampleDaoImpl.**

Contiennent les dao permettant de manupuler la BDD.

- **opuslib.module.base.servlet**

Contient les servlets.

- **opuslib.model**

Contient le modèle utilisateur permettant de récupérer les informations de l'utilisateur connecté et le modèle Mail permettant l'envoi de mail.

- **opuslib.core.tools** **ApplicationContext et ApplicationContextBean obligatoires.**

Permettent de récupérer les informations de connexion à la BDD de l'instance d'Opuslib **(Les deux classes doivent être absolument identiques à celles du cœur d'Opuslib).**

Les classes **ApplicationMailContext** et **ApplicationMailContextBean** permettent de récupérer le login et le mot de passe de l'adresse gmail du serveur du cœur d'Opuslib dans le cas où vous avez besoin d'envoyer des mails (voir cette partie).

Fonctionnement d'un module

Les servlets qui vont être présentées maintenant sont **nécessaires** au bon fonctionnement d'un module. Elles sont dans le package **opuslib.module.base.boot**.

1. Servlet On

Servlet appelée lors du déploiement d'un module dans l'instance d'Opuslib.

Elle récupère l'id du module renvoyé par le cœur et exécute sur celui-ci l'installation d'un module, c'est-à-dire:

- l'ajout des liens entre les pages du modules et du cœur (voir exemple ci-dessous)
- l'ajout des clés d'appels des servlets du module sur le cœur (voir exemple ci-dessous)
- l'ajout des clés de la map de langue sur le coeur (voir exemple ci-dessous)
- l'ajout de la version du module.

2. Servlet Off

Servlet appelée lors de la suppression d'un module dans l'instance d'Opuslib.

Elle récupère aussi l'id du module renvoyé par le cœur et exécute les opérations suivantes :

- la suppression des liens entre les pages du modules et du cœur
- la suppression des clés d'appels des servlets du module sur le cœur
- la suppression des clés de la map de langue sur le coeur
- la suppression de l'historique des versions déployées du module.

3. Servlet Activate

Servlet appelée lors de l'activation d'un module dans l'instance d'Opuslib.

Elle active sur le cœur le module.

4. Servlet Pause

Servlet appelée lors de la pause d'un module dans l'instance d'Opuslib.

Elle met en pause sur le cœur le module.

5. Servlet Update

Servlet appelée lors de la mise à jour d'un module dans l'instance d'Opuslib.

Dans notre module de base, elle met à jour sa version.

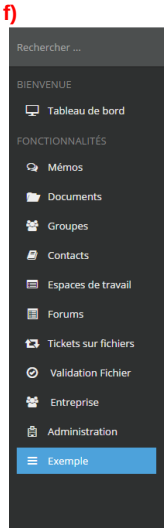
6. Module dispatcher du cœur (opuslib.core.ged.ModuleDispatcherServlet) et filtre du module (opuslib.common.filter.SessionFilter)

Le module dispatcher du coeur permet, lors de l'affichage d'une page d'un module, de récupérer le menu de gauche, le header et le footer, éléments communs à toutes les pages d'Opuslib.

Par conséquent, lorsque l'on créé une page pour un module, il n'est pas nécessaire d'ajouter les balises <*body*> ou <*html*> mais seulement le contenu de la page (voir exemple ci-dessous).

Lorsque l'on veut appeler une page de notre module, il faut donc appeler la servlet **/modules** avec comme paramètres :

- **_name=<key>** avec **key** la clé pointant vers la servlet appelant notre page de module **(obligatoire)**
- **_menu=<item>** avec **item** la clé représentant l'item du menu étant en classe **active** (élément surligné en bleu ci-dessous) **(facultatif)**



Avec le paramètre **_name**, le module dispatcher récupère en BDD la servlet à appeler et l'appelle via un **<c:import>** avec en paramètres (**<c:param>**) l'id, l'UUID de l'utilisateur connecté ainsi que sa locale utilisée. L'id et l'UUID de l'utilisateur connecté permettent de vérifier que c'est bien un utilisateur d'Opuslib qui appelle notre module Ces trois paramètres sont récupérés dans le filtre de notre module et mis en session.

opuslib.common.filter.SessionFilter

```
String locale = request.getParameter("_opus_user_locale");
  String userId = request.getParameter("_opus_user_id") != null ?
request.getParameter("_opus_user_id"):(String)request.getAttribute("user
Id");
  String uuid = request.getParameter("_opus_user_uuid") != null ?
request.getParameter("_opus_user_uuid"):(String)request.getAttribute("uu
id");
  String menu = request.getParameter("_menu");
  if (userId != null){
   request.setAttribute("userId", userId);
  }
  if (langue != null){
    request.setAttribute("langueMap", langue);
  }
  if (cellId != null){
   request.setAttribute("cellId", cellId);
  }
  if (menu != null){
   request.setAttribute("menu", menu);
  }
  if (uuid !=null){
   request.setAttribute("uuid", uuid);
  }
```

De plus, le module dispatcher rajoute dans l'url d'appel de notre servlet de module tous les paramètres qui lui sont passés.

> **Exemple:** un appel de **/modules?_name=Example.page1&_menu=base&spaceId=3** implique que notre servlet sera appelée de cette manière : **/example?spaceId=3&_menu=base**.

> Toutes les servlets appelées avec notre module le sont en **GET**.

7. Envoi de mails avec l'API de mail

Le cas échéant vous pouvez appeler l'API d'envoi de mails en POST avec le lien suivant : **url_de_l'application/api/mail**.

La requête POST doit correspondre aux critères suivants :

- pas de header
- le body est un JSON aux attributs suivants :
    - **mailDestinataire** : le mail du destinataire
    - **subject** : le sujet
    - **message** : le contenu du mail (texte simple ou html).

Une fois appelée avec le JSON dans le body, l'API répond par un *Mail sended* si le mail est envoyé avec succès ou *Mail not sended* dans le cas contraire.

> Le module contient un formulaire sur la page pageExample.jsp.qui appelle la servletSendMailde notre module. Celle-ci envoie une requête POST en Java à notre API de mails.

Insertion d’un item dans le menu d’Opuslib

1. Création de la jsp buttonMenu.jsp

buttonMenu.jsp

```
<%@ page language="java" session="false" contentType="text/html;
charset=utf-8" pageEncoding="utf-8"%>
<script>
 $(function() {
  var trad = '${langueMap["base.menu"]}';
  var menu = '${menu}';
  if (menu == 'base'){
   $('.sidebar-nav-menu').append("<li><a id='menu-base'
href='modules?_name=Example.page1&_menu=base' class='active'><i
class='fa fa-bars'></i>"+trad+"</a></li>");
  }else{
   $('.sidebar-nav-menu').append("<li><a id='menu-base'
href='modules?_name=Example.page1&_menu=base'><i class='fa
fa-bars'></i>"+trad+"</a></li>");
  }

 });
</script>
```

On insère dans la sidebar en javascript, un lien qui appelle le module dispatcher du cœur avec comme paramètre _***name=Example.page1*** q ui est la clé pointant vers le chemin de la servlet appelée (***/Base/example***) et _**menu=base** qui représente le menu avec la classe **active**.

Voir détails ici.

2. Référencement de la .jsp dans la BDD d’Opuslib

opuslib.module.base.dao.impl.CellDaoImpl

```
@Override
    public void init(String idCell){
        Connection connection = null;
        Statement statement = null;
        try {
            Class.forName(CLASSFORNAME);
            connection = DriverManager.getConnection(URL, UTILISATEUR,
MOTDEPASSE);
            statement = connection.createStatement();
            statement.addBatch("INSERT INTO
Cell_source(id,file,page,rank,idCell) VALUES"
                    +
"(NULL,'buttonMenu.jsp','sidebarLeft.jsp',1,"+idCell+")");
            statement.executeBatch();

        } catch (SQLException | ClassNotFoundException e) {
            e.printStackTrace();
        } finally {
            closeConnection(connection);
        }
    }
```

On insère dans la table **Cell_source** les pages que l'on veut faire apparaître dans le cœur. **File** correspond à la page du module que l'on veut insérer, ici, **buttonMenu.jsp**. **Page** correspond à la page du cœur où l'insertion sera faite, ici, **sidebarLeft.jsp**.

3. Ajout du texte en multilingue

opuslib.module.base.dao.impl.CellDaoImpl

```
@Override
 public void insertLangue(){
  Connection connection = null;
  Statement statement = null;
  try {
   Class.forName(CLASSFORNAME);
   connection = DriverManager.getConnection(URL, UTILISATEUR,
MOTDEPASSE);
   statement = connection.createStatement();
   statement.addBatch("INSERT INTO Langues(id,object,fr,gb)
VALUES(NULL,'base.menu','Exemple','Example')");

   statement.executeBatch();

  } catch (SQLException | ClassNotFoundException e) {
   e.printStackTrace();
  } finally {
   closeConnection(connection);
  }
 }
```

On insère dans la table **Langues** du coeur d'Opuslib les clés de la langueMap utilisées, dans notre cas **base.menu** (voir code 1. Création de la jsp buttonMenu.jsp). Ces clés ainsi que leurs valeurs sont récupérées dans notre filtre (voir ci-dessus).

Insertion d'une nouvelle page .jsp

1. Création de la pageExample.jsp

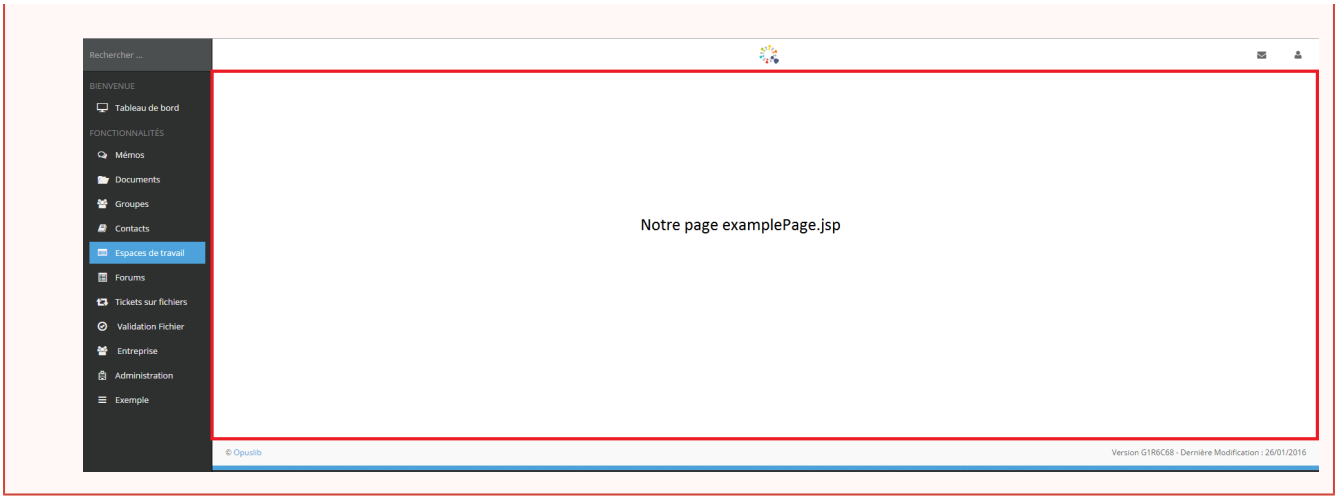
pageExample.jsp

```
<%@ page language="java" session="false" contentType="text/html;
charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<!-- Page content -->
<div id="page-content" class="block">
    <!-- Dashboard Header -->
    <div class="block-header">
        <div class="row remove-margin">
            <!-- Title -->
            <div class="col-md-4">
                <!-- If you do not want a link in the header, instead of
.header-title-link you can use a div with the class .header-section -->
                <a href="" class="header-title-link">
                    <h1><i
class="animation-expandUp"></i>${langueMap['base.title']}</h1>
                </a>
            </div>
            <!-- END Title -->
            <div class="col-md-8">
                <!-- Outer Grid -->
                <div class="row">
                 <div class="col-sm-6">
     <!-- Inner Grid 1 -->
      <div class="row">

       <div class="col-xs-6">
        <a href="getHome"class="header-link" >
         <h1 class="animation-pullDown">
          <i class="glyphicon-unshare animation-expandUp"></i><br>
          <small>${langueMap['base.backIndex']}</small>
         </h1>
        </a>
       </div>
      </div>
      <!-- END Inner Grid 1 -->
     </div>
            </div>
            <!-- END Outer Grid  -->
           </div>
        </div>
    </div>


    <h2>${message }</h2>
</div>
<!-- END Page Content -->
```

Pas besoin de mettre de balise <html> <body> etc... Il faut voir notre page de module comme partie intégrante d'une page du cœur comme ci-dessous.



2. Intégration de cette page dans le coeur Opuslib

1. Pour l'accès à nos pages créées par notre module, on passe par la servlet **/modules** présente dans le cœur avec comme paramètre ***_name=Example.page1*** qui est la clé pointant vers le chemin de la servlet appelée (***/Base/example***). Servlet qui, elle, renvoie vers notre page **pageExample.jsp**.

   > Voir l'appel de la servlet **/modules** 1. Page buttonMenu.jsp

2. Ensuite, on déclare en BDD la valeur de la clé **Example.page1** :

opuslib.module.base.dao.impl.CellDaoImpl

```
@Override
 public void pageRegister(String idCell){
  Connection connection = null;
  Statement statement = null;
  try {
   Class.forName(CLASSFORNAME);
   connection = DriverManager.getConnection(URL, UTILISATEUR,
MOTDEPASSE);
   statement = connection.createStatement();
   statement.addBatch("INSERT INTO cell_link(id,name,url,idCell)
VALUES(NULL,'Example.page1','/Base/example',"+idCell+")");
   statement.executeBatch();

  } catch (SQLException | ClassNotFoundException e) {
   e.printStackTrace();
  } finally {
   closeConnection(connection);
  }
 }
```

La valeur de **name** correspond à notre clé, la valeur **url** correspond à l'url de la servlet de notre module. **Veuillez à bien insérer l'url de notre module qui correspond au nom du module (voir dernière partie de ce document) ici /Base avant l'url propre de la servlet.**

3. Ajout du texte en multilingue

Voir cette partie 3.Ajout **du texte en multilingue.**

Intégration d'un bouton dans le cœur d'Opuslib (la liste des espaces)

1. Création de la page buttonSpacesExample.jsp

buttonSpacesExample.jsp

```
<%@ page language="java" session="false" contentType="text/html;
charset=utf-8" pageEncoding="utf-8"%>
<script>
 $(function() {


  $('.filesList').each(function() {
   var id= this.id.split("_")[0];
   var trad = "${langueMap['base.button']}";
   $("#"+id+"_files").prepend("<a
href='modules?_name=Example.page1&_menu=base&spaceId="+id+"' class='btn
btn-info btn-sm' style='margin-bottom: 10px'>"+trad+"</a>");
  });


 });
</script>
```

On parcourt chaque bloc espace comme ci-dessus afin de lui ajouter un bouton qui renvoie vers notre page **pageExample.jsp** avec comme paramètre un id de l'espace qui nous permettra d'afficher le nom de l'espace sur celle-ci. Comme pour la partie précédente, on fait appel au module dispatcher du cœur **/modules** pour afficher notre page.

2. Ajout du texte en multilingue

Voir cette partie 3.Ajout **du texte en multilingue.**

3. Référencement de la .jsp en BDD

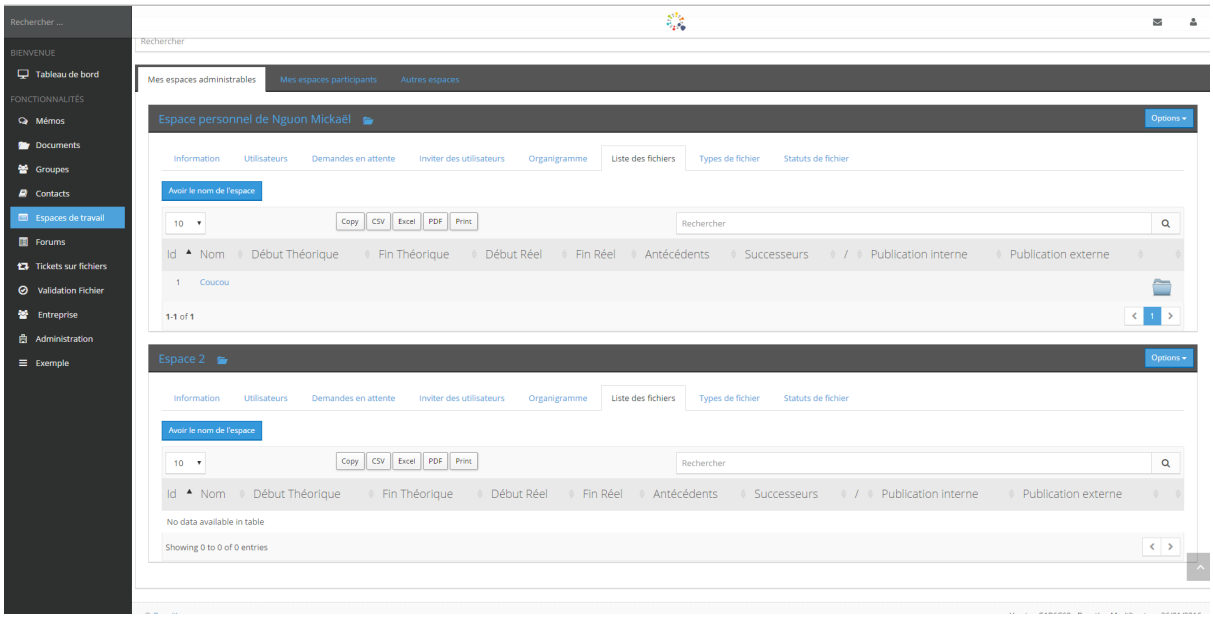
opuslib.module.base.dao.impl.CellDaoImpl

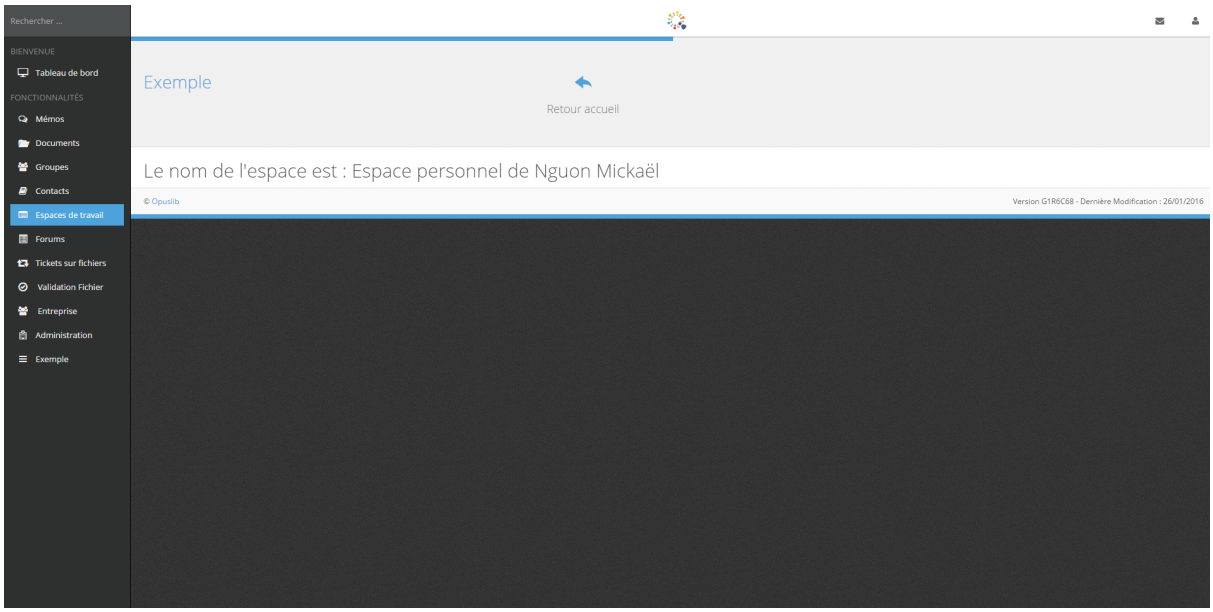
```
@Override
 public void init(String idCell){
  Connection connection = null;
  Statement statement = null;
  try {
   Class.forName(CLASSFORNAME);
   connection = DriverManager.getConnection(URL, UTILISATEUR,
MOTDEPASSE);
   statement = connection.createStatement();
   statement.addBatch("INSERT INTO Cell_source(id,file,page,rank,idCell)
VALUES"
     + "(NULL,'buttonSpacesExample.jsp','spaces.jsp',1,"+idCell+")");
   statement.addBatch("INSERT INTO Cell_source(id,file,page,rank,idCell)
VALUES"
     + "(NULL,'buttonSpacesExample.jsp','space.jsp',1,"+idCell+")");
   statement.executeBatch();

  } catch (SQLException | ClassNotFoundException e) {
   e.printStackTrace();
  } finally {
   closeConnection(connection);
  }
 }
```

Ici comme pour cette partie, on indique que notre page doit être intégrée dans les pages **spaces.jsp** et **space.jsp** du coeur.

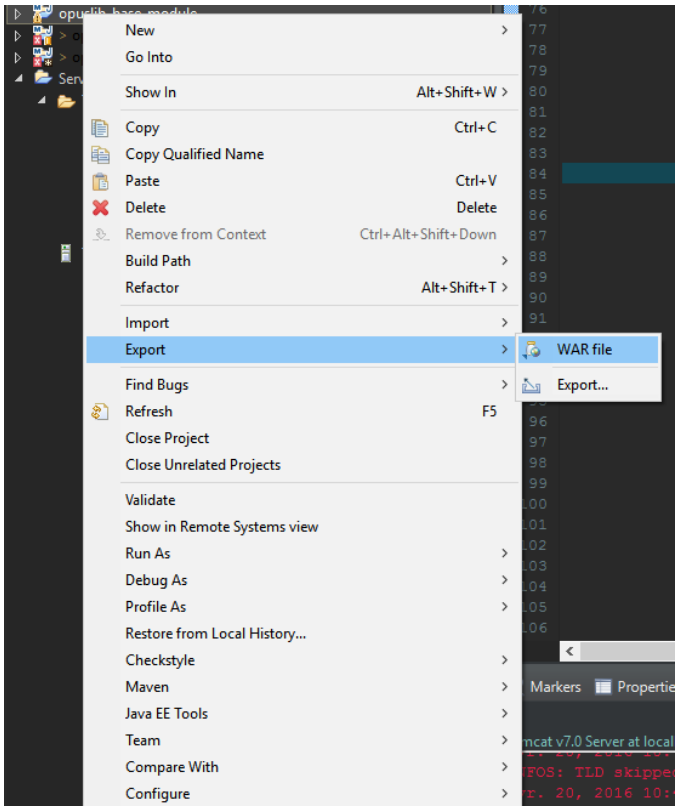
4. Résultat

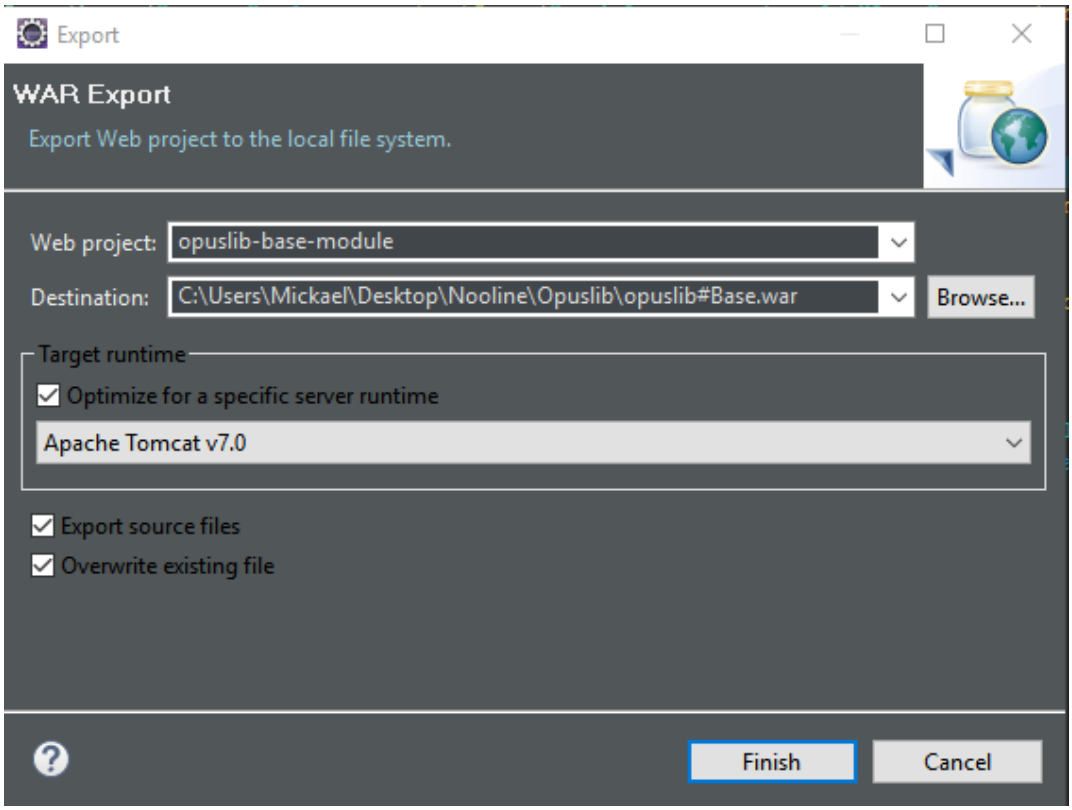




Activation du module sur l'instance d'Opuslib

1. Export en WAR de notre module Base



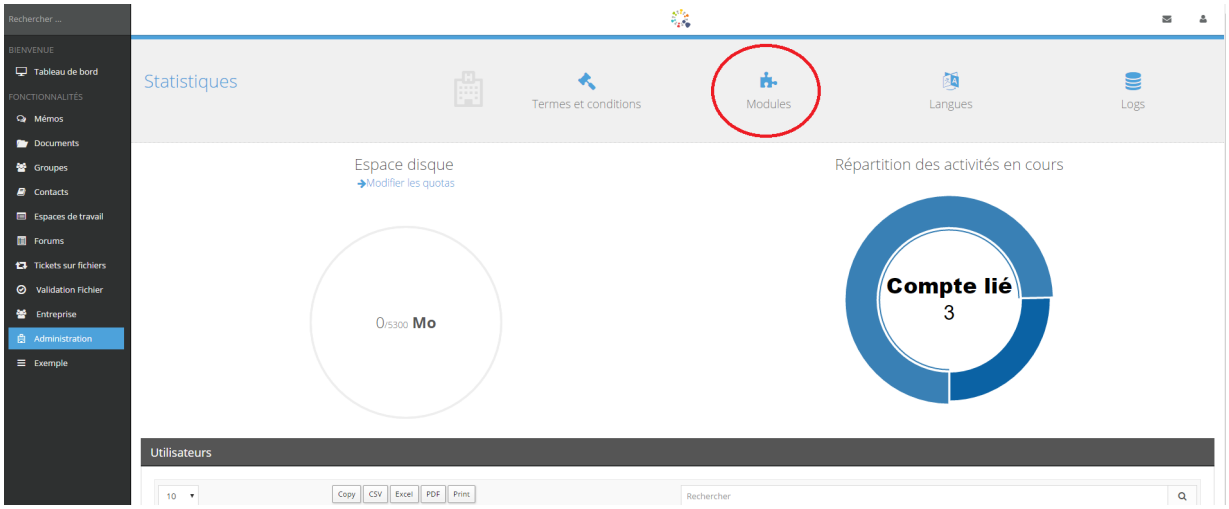


Votre module doit absolument avoir comme nom suivant : *Contexte_web_de_l'instance_d'Opuslib#Nom_du_module.war*. Exemple : *opuslib#Base.war.*

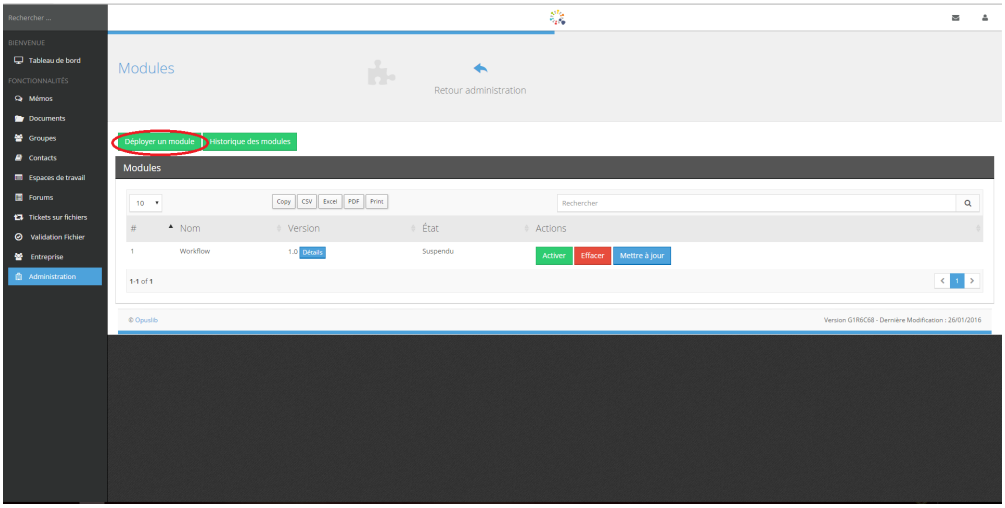
2. Déploiement du module sur l'instance d'Opuslib

Prérequis : être admin sur Opuslib.

Se rendre dans la partie administration d'Opuslib puis cliquer sur Modules en haut de la page comme ci-dessous



Une fois sur la page des modules et cliquer sur **Déployer un module** comme indiqué ci-dessous :



Ensuite sélectionner le module à déployer et valider. Vous pouvez ensuite **Activer votre module**.

Votre module est maintenant accessible à tous les utilisateurs d'Opuslib.